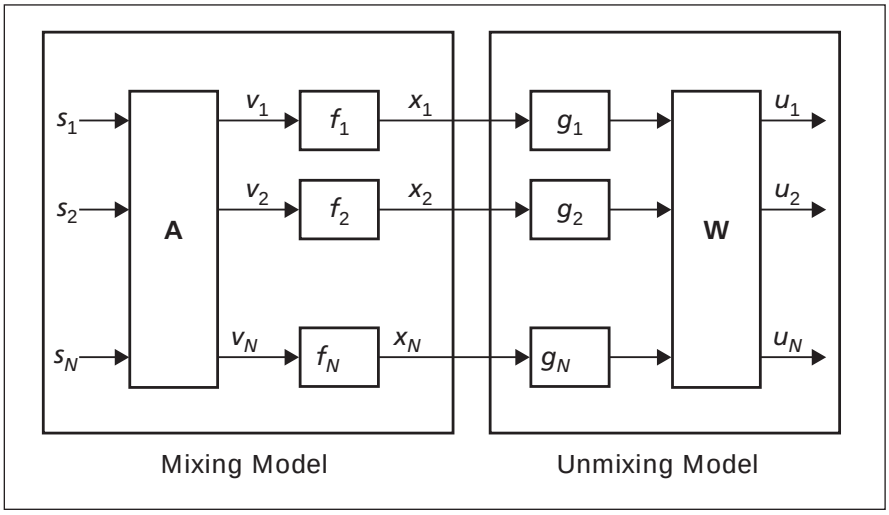


Mixing and Distortion Operations and their inverses are represented in these block-diagram representations of mixing and unmixing models.

nals. For the ith mixture signal, the distorted signal is given by $x_i = f_i(v_i)$, where f_i is one of the initially unknown nonlinear functions. Thus, the vector

$$\mathbf{x}(n) = [x_1(n),x_2(n),...x_N(n)]^T$$

represents instrumentation signals presented for analysis. The distortion in signal x_i is removed by means of a corresponding initially unknown inverse nonlinear function g_i. Finally, the signals are unmixed by means of initially unknown matrix **W** to obtain output vector

$$\mathbf{u}(n) = [u_1(n),u_2(n),...u_N(n)]^T.$$

In the ideal case, **W** would be the inverse of **A** and the output vector **u** would equal the vector, **s**, of source signals.

The particular nonlinear ICA problem is to calculate the nonlinear inverse functions g_i and matrix **W** such that **u** calculated by use of them is a close approximation of **s**. For the purpose of the present algorithm for solving this problem, it is assumed that the inverse nonlinear functions g_i are smooth and can be approximated by polynomials. The algorithm finds the components of the unmixing matrix **W** and the coefficients of the polynomial approximations of g_i by a gradient-descent method. This algorithm utilizes the kurtosis of the components of the output vector **u** as an objective function (in effect, an error measure) that it seeks to minimize. In using the kurtosis, this algorithm stands in contrast to prior algorithms that utilize other objective functions, including statistical functions other than the kurtosis.

This work was done by Vu Duong and Allen Stubberud of Caltech for NASA's Jet Propulsion Laboratory.

In accordance with Public Law 96-517, the contractor has elected to retain title to this invention. Inquiries concerning rights for its commercial use should be addressed to:

Innovative Technology Assets Management
JPL
Mail Stop 202-233
4800 Oak Grove Drive
Pasadena, CA 91109-8099
(818) 354-2240
E-mail: iaoffice@jpl.nasa.gov

Refer to NPO-43088, volume and number of this NASA Tech Briefs issue, *and the page number.*

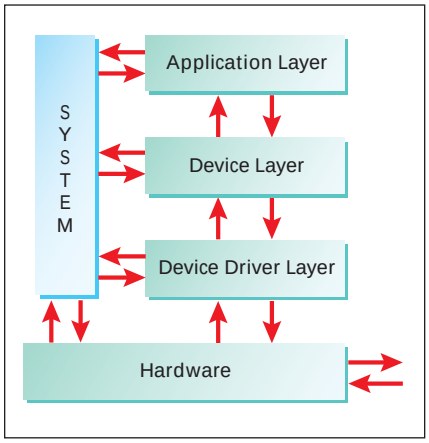
Robust Software Architecture for Robots

Generalized software can be readily tailored for specific applications.

NASA's Jet Propulsion Laboratory, Pasadena, California

"Robust Real-Time Reconfigurable Robotics Software Architecture" ("R4SA") is the name of both a software architecture and software that embodies the architecture. The architecture was conceived in the spirit of current practice in designing modular, hard, real-time aerospace systems. The architecture facilitates the integration of new sensory, motor, and control software modules into the software of a given robotic system. R4SA was developed for initial application aboard exploratory mobile robots on Mars, but is adaptable to terrestrial robotic systems, real-time embedded computing systems in general, and robotic toys.

The R4SA software, written in clean ANSI C, establishes an onboard, real-time computing environment. The R4SA architecture features three layers: The lowest is the device-driver layer, the highest is the application layer, and the device layer lies at the middle (see figure).



The **R4SA Architecture** features three levels corresponding to different levels of abstraction.

The device-driver layer handles all hardware dependencies. It completely hides the details of how a device works. Activities directed by users are performed by means of well-defined interfaces. Each type of device driver is equipped with its own well-defined interface. For example, the device-driver interface for an analog-to-digital converter differs from that for a digital-to-analog converter.

The device layer provides the means for abstracting the high-level software in the application layer from the hardware dependencies. The device layer provides all motion-control computations, including those for general proportional + integral + derivative controllers, profilers, controllers for such mechanical components as wheels and arms, coordinate-system transformations for odometry and inertial navigation, vision processing, instrument interfaces, communication among multiple robots, and kinematics for a multiple-wheel or multiple-leg robot.

The application layer provides application programs that a robot can execute. Examples of application programs include those needed to perform such prescribed maneuvers as avoiding obstacles while moving from a specified starting point to a specified goal point or turning a robot in place through a specified azimuthal angle. Each robot is provided with application software representing its own unique set of commands. The software establishes a graphical user interface (GUI) for exchanging command information with external computing systems. Via the GUI and its supporting interface software, a user can select and assemble, from the aforementioned set, commands appropriate to the task at hand and send the commands to the robot for execution. System software that interacts with the R4SA software at all three levels establishes a synchronized control environment.

The R4SA software features two modes of execution: before real time (BRT) and real time (RT). In the BRT mode, a text configuration file is read in (each robot has its own unique file) and then device-driver-layer, device-layer, and application-layer initialization functions are executed. If execution is successful, then the system jumps into the RT mode, in which the system is ready to receive and execute commands.

One goal in developing the R4SA architecture was to provide one computer code for many robots. The unique executable code for each robot is built by use of a configuration feature file. The set of features for a given robot is selected from a feature database on the basis of the hardware and mechanical capabilities of that robot. Recompilation of code is straightforward: modifications can readily be performed in the field by use of simple laptop-computer development and debugging software tools.

This work was done by Hrand Aghazarian, Eric Baumgartner, and Michael Garrett of Caltech for NASA's Jet Propulsion Laboratory. Further information is contained in a TSP (see page 1).

The software used in this innovation is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-41796.

R4SA for Controlling Robots

NASA's Jet Propulsion Laboratory, Pasadena, California

The R4SA GUI mentioned in the immediately preceding article is a user-friendly interface for controlling one or more robot(s). This GUI makes it possible to perform meaningful real-time field experiments and research in robotics at an unmatched level of fidelity, within minutes of setup. It provides such powerful graphing modes as that of a digitizing oscilloscope that displays up to 250 variables at rates between 1 and 200 Hz. This GUI can be configured as multiple intuitive interfaces for acquisition of data, command, and control to enable rapid testing of subsystems or an entire robot system while simultaneously performing analysis of data.

The R4SA software establishes an intuitive component-based design environment that can be easily reconfigured for any robotic platform by creating or editing setup configuration files. The R4SA GUI enables event-driven and conditional sequencing similar to those of Mars Exploration Rover (MER) operations. It has been certified as part of the MER ground support equipment and, therefore, is allowed to be utilized in conjunction with MER flight hardware. The R4SA GUI could also be adapted to use in embedded computing systems, other than that of the MER, for commanding and real-time analysis of data.

This work was done by Hrand Aghazarian of Caltech for NASA's Jet Propulsion Laboratory. Further information is contained in a TSP (see page 1).

The software used in this innovation is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-41797.

Bio-Inspired Neural Model for Learning Dynamic Models

This model could be a basis for fast speech- and image-recognition computers.

NASA's Jet Propulsion Laboratory, Pasadena, California

A neural-network mathematical model that, relative to prior such models, places greater emphasis on some of the temporal aspects of real neural physical processes, has been proposed as a basis for massively parallel, distributed algorithms that learn dynamic models of possibly complex external processes by means of learning rules that are local in space and time. The algorithms could be made to perform such functions as recognition and prediction of words in speech and of objects depicted in video images. The approach embodied in this model is said to be "hardware-friendly" in the following sense: The algorithms would be amenable to execution by special-purpose computers implemented as very-large-scale integrated (VLSI) circuits that would operate at relatively high speeds and low power demands.

It is necessary to present a large amount of background information to give meaning to a brief summary of the present neural-network model:

- A dynamic model to be learned by the present neural-network model is of a type denoted an internal model or predictor. In simplest terms, an internal model is a set of equations that predicts future measurements on the basis of past and current ones. Internal models have been used in controlling industrial plants and machines (including robots).
- One of the conclusions drawn from Pavlov's famous experiments was the observation that reinforcers of learning (basically, rewards and punishments) become progressively less efficient for causing adaptation of